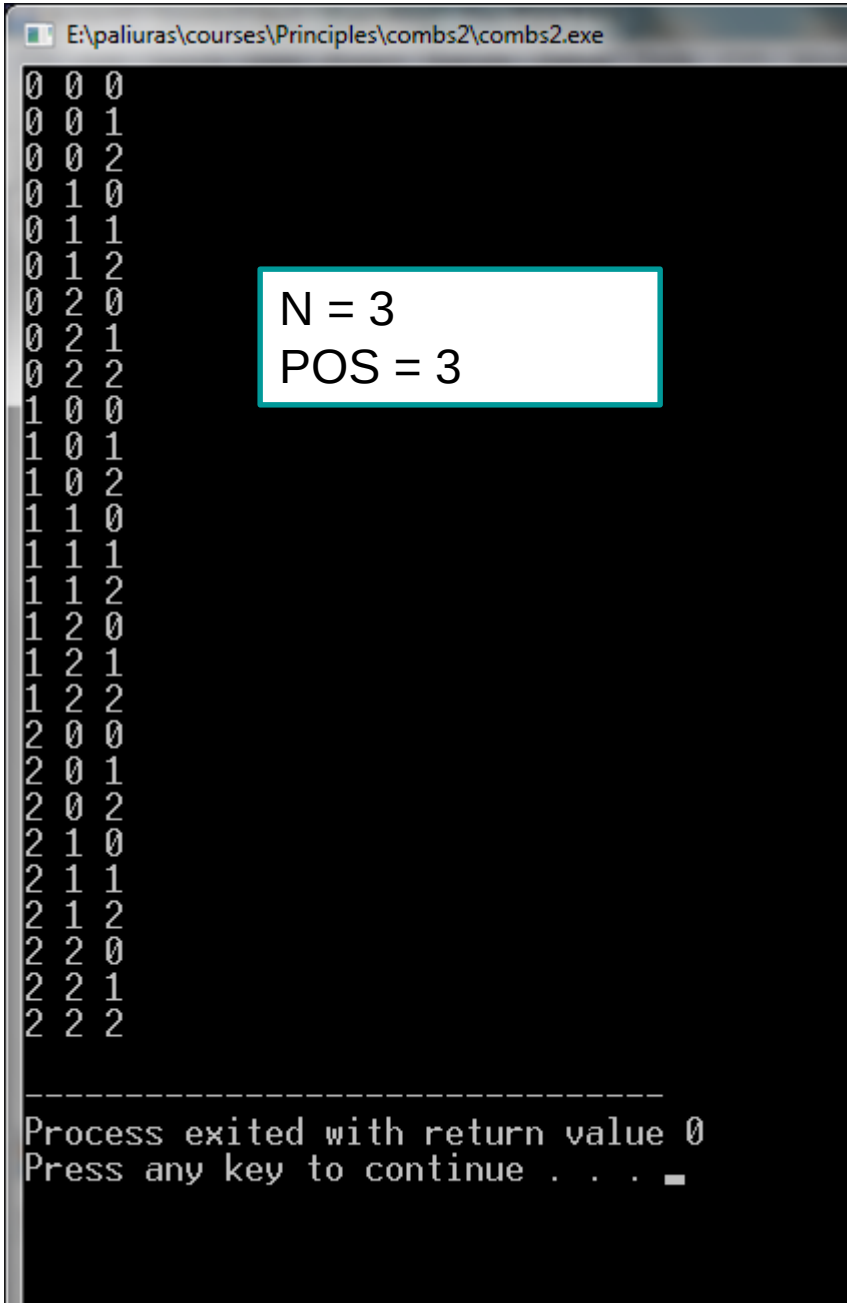
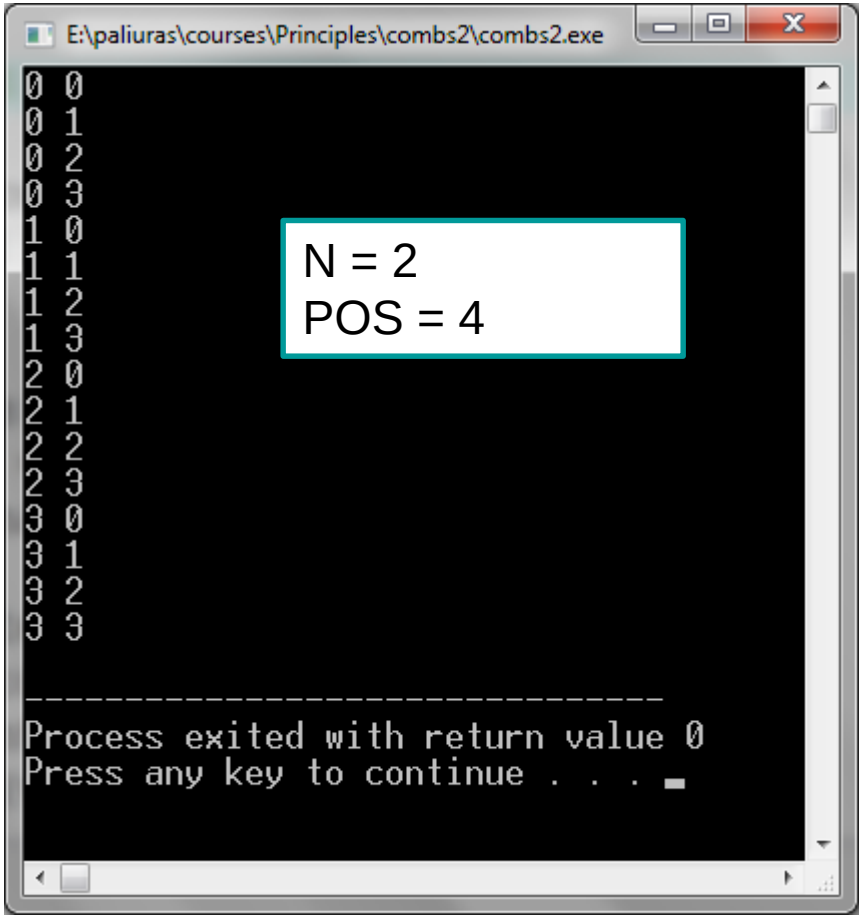
# *Διαδικαστικός Προγραμματισμός*

Βασίλης Παλιουράς

# Παράδειγμα

- Να γραφεί ένα πρόγραμμα που να δημιουργεί και να εμφανίζει όλους τους συνδυασμούς Ν ακεραίων, καθένας από τους οποίους μπορεί να πάρει POS τιμές.


E:\paliuras\courses\Principles\combs2\combs2.exe
0 0 0
0 0 1
0 0 2
0 1 0
0 1 1
0 1 2
0 2 0
0 2 1
0 2 2
1 0 0
1 0 1
1 0 2
1 1 0
1 1 1
1 1 2
1 2 0
1 2 1
1 2 2
2 0 0
2 0 1
2 0 2
2 1 0
2 1 1
2 1 2
2 2 0
2 2 1
2 2 2
Process exited with return value 0
Press any key to continue . . .
N = 3
POS = 3


E:\paliuras\courses\Principles\combs2\combs2.exe
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
3 0
3 1
3 2
3 3
Process exited with return value 0
Press any key to continue . . .
N = 2
POS = 4

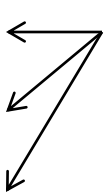
```c
#include <stdio.h>
#define POS 3
#define N 3
void report (const int [N]);

int main() {
        int values[3];

        for (values[2]=0; values[2] < POS; values[2]++)
                for (values[1]=0; values[1] < POS; values[1]++)
                        for (values[0]=0; values[0]< POS; values[0]++)
                        report(values);

        return 0;
}

void report (const int v[N]) {
  int i;

  for (i=N-1;i>=0;i--)
    printf("%d ",v[i]);

  printf("\n");

}
```

Όμως ο αριθμός των for μέσα σε for (nested fors) είναι σταθερός!

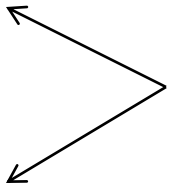
=>

Χρειάζεται άλλη μέθοδος.

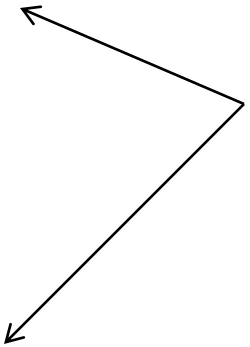
- 000
- 001
- 002
- 010
- 011
- 012
- 020
- 021
- 022
- 100
- 101
- 102
- …
- 122
- 200
- …
- 221
- 222



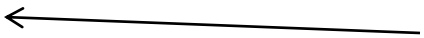
Αύξησε το τελευταίο
αν γίνεται



Αν δεν γίνεται μηδένισέ το και αύξησε το επόμενο αν γίνεται



…αν δεν γίνεται να αυξήσεις το επόμενο, μηδένισέ το
και αύξησε το μεθεπόμενο, αν γίνεται



Πότε τελειώνει η επανάληψη;;

```c
#include <stdio.h>
#define N 5
#define COUNT 32
#define POS 2
void report (const int [N]);
void increaseall(int [N]);

int main() {
  int values[N] = {0};
  int count = 0;
  while (count < COUNT ) {
        report(values);
        count ++;
        increaseall(values);
    }
  return 0;
}
void report (const int v[N]) {
  int i;
  for (i=N-1;i>=0;i--)
    printf("%d ",v[i]);
  printf("\n");
}
```

Εμφάνισε το συνδυασμό

Αύξησε μετρητή
(ξέρεις ότι χρειάζονται $POS^N$)

Προχώρησε στον επόμενο συνδυασμό

```c
void increaseall(int v[N]) {
        int j;
        for (j=0; j<N; j++) {
            if (v[j]<POS-1) {
              v[j] ++;
              return ;
            }
            else
              v[j] = 0;
        }
}
```

Αν γίνεται ,
αύξησε τη θέση και
επέστρεψε στο main

Αλλιώς μηδένισε τη θέση
Και συνέχισε στο βρόχο

# Απλούστερη λύση (χωρίς μετρητή)

```c
#include <stdio.h>
#define N 2
#define POS  4
void report (const int [N]);
int increaseall(int [N]);
int main() {
        int values[N] = {0};
        int terminate = 0;
        while ( !terminate ) {
          report(values);
          terminate = increaseall(values);
    }
    return 0;
}
void report (const int v[N]) {
  int i;
  for (i=N-1;i>=0;i--)
    printf("%d ",v[i]);
  printf("\n");
}
```

Αλλάζουμε τον τύπο της increaseall από **void** Σε **int**

```c
int increaseall(int v[N]) {
        int j;
        for (j=0; j<N; j++) {
            if (v[j] < POS-1) {
               v[j] ++;
               return 0;
            }
            else
               v[j] = 0;
        }
        return 1;
}
```

Σε περίπτωση που αυξήσει κάποια θέση επιστρέφει 0

Σε περίπτωση που δεν υπάρχουν άλλα ψηφία, (άρα τέλος) επιστρέφει 1

# Άλλες υλοποιήσεις της main

```
int main() {
        int values[N] = {0};
        int terminate = 0;
        for (;!terminate;terminate=increaseall(values))
                report(values);
        return 0;
}
```

```
int main() {
        int values[N] = {0};
        do {
          report(values);
        } while (!increaseall(values));
        return 0;
}
```

# Άλλες υλοποιήσεις της increaseall

```
int increaseall(int v[N]) {
        int j;

        for (j=0; j<N && v[j]==POS-1; j++)
                v[j] = 0;
        if (j<N) {v[j]++;
                  return 0;
                 }
        return 1;
}
```

```
int increaseall(int v[N]) {
        int j=0, condition;
        for (; j<N && v[j]==POS-1; v[j++]=0);
        if (condition = j<N) v[j] += condition;
        return !(condition);
}
```

# Για δυαδικούς αριθμούς

```
#include <stdio.h>

int nbit(int x, int n) {

        return  (x & (1<<n) ) != 0 ;

}

int main(void) {

        int i;
        int n = 4;
        int bit;

        for (i=0;i<16; i++ ) {
                for (bit = n - 1; bit >=0; bit -- ) {
                        printf("%d ", nbit(i, bit));
                }
                printf("\n");
        }

        return 0;
}
```

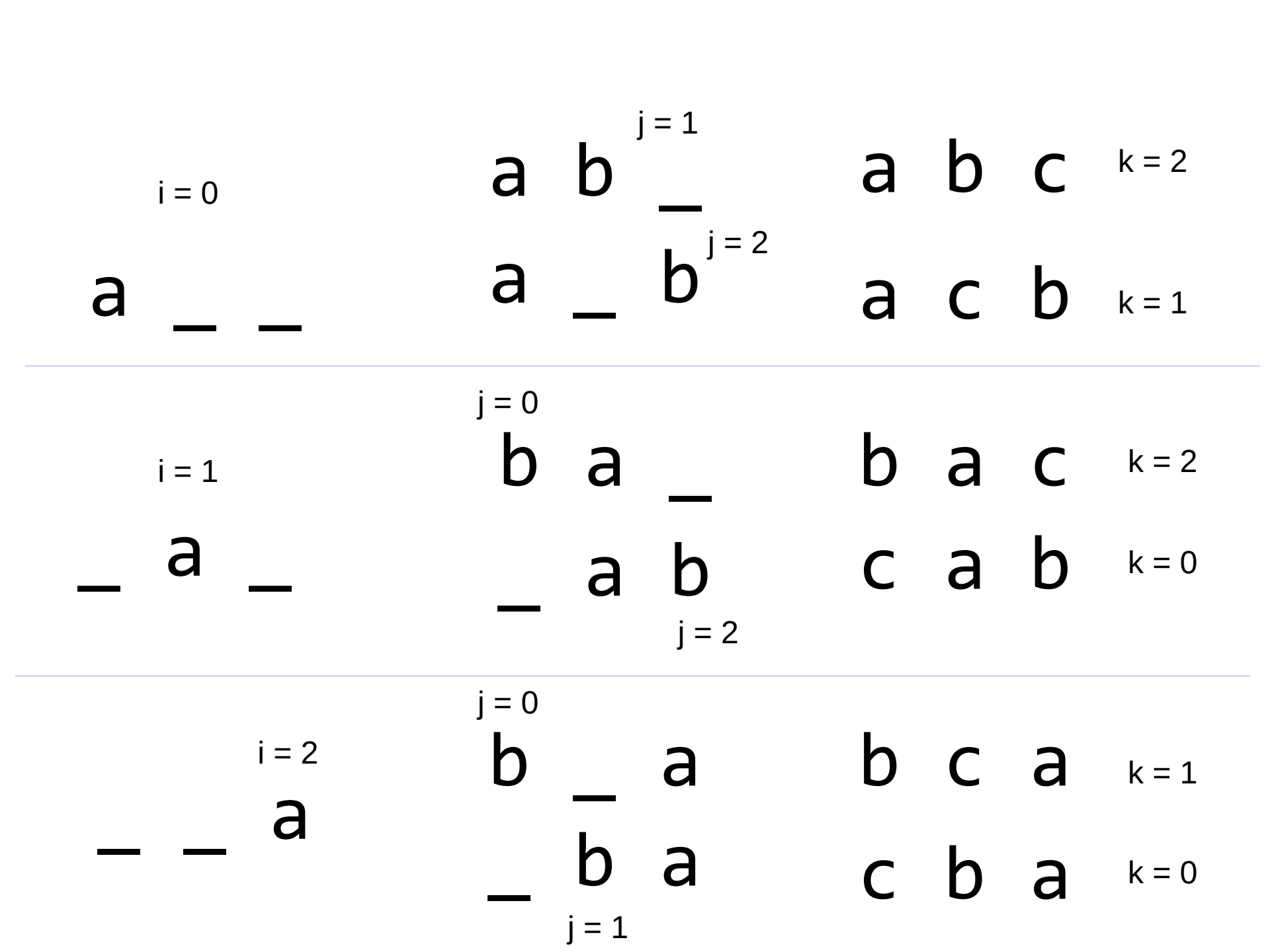
```c
#include <stdio.h>

int nbit(int x, int n) {

        return  (x & (1<<n) ) != 0 ;

}

int main(void) {

        int i;
        int n = 4;
        int bit;
        int combs = 1 << n;

        for (i=0;i< combs ; i++ ) {
                for (bit = n - 1; bit >=0; bit -- ) {
                        printf("%d ", nbit(i, bit));
                }
                printf("\n");
        }

        return 0;
}
```

```c
#include <stdio.h>

int nbit(int x, int n) {
          return  (x & (1<<n) ) != 0 ;
}

int getbinvalues(int * bits, int i, int n) {
          int bit;
          for (bit = n - 1; bit >=0; bit -- )        {
            bits[bit] = nbit(i, bit);
          }
          return 0 ;
}

int report(int * bits, int n){
          int bit;
          for (bit = n -1; bit >=0 ; bit--)
                    printf("%d ", bits[bit]);
          printf("\n");
          return 0;
}

int main(void) {
 int i;
 int n = 4;
 int combs = 1 << n;
 int bits[4];

 for (i=0;i<combs; i++ ) {
      getbinvalues(bits, i, n);
      report(bits,n);
      }

 return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
int nbit(int x, int n) {
          return  (x & (1<<n) ) != 0 ;
}

int getbinvalues(int * bits, int i, int n) {
   int bit;
   for (bit = n - 1; bit >=0; bit -- ) {
             bits[bit] = nbit(i, bit);
   }
   return 0;
}

int report(int * bits, int n) {
   int bit;
   for (bit = n -1; bit >=0 ; bit--)
       printf("%d ", bits[bit]);
   printf("\n");
   return 0;
}
int main(void) {
    int i;
    int n = 4;
    int combs = 1 << n;
    int * bits;
    bits = malloc (n * sizeof (int));
    for (i=0;i<combs; i++ ) {
        getbinvalues(bits, i, n);
        report(bits,n);
    }

    free(bits);
    return 0;
}
```

- Να κατασκευαστεί συνάρτηση C90 η οποία να τυπώνει τις αναδιατάξεις των χαρακτήρων ενός αλφαριθμητικού το οποίο δίνεται ως παράμετρος.

- Πρώτα για τρεις χαρακτήρες
- Μετά γενικεύουμε


i = 0
a _ _
j = 1
a b _
k = 2
a b c
j = 2
a _ b
k = 1
a c b
i = 1
_ a _
j = 0
b a _
k = 2
b a c
_ a b
k = 0
c a b
j = 2
i = 2
_ _ a
j = 0
b _ a
k = 1
b c a
_ b a
k = 0
c b a
j = 1

```
int permute(char w[]) {
        char *temp;
        int i, j, k;
        temp = calloc (strlen(w)+1, sizeof(char));

        for (i=0;i<3; i++) {
                temp[i] = w[0];
                for (j = 0 ; j<3; j++) {
                        if (j != i ) {
                                temp[j] = w[1];
                                for (k=0; k<3; k++) {
                                        if (k!= i && k!=j ) {
                                                temp[k] = w[2];
                                                printf("%s\n", temp);
                                        }
                                }
                        }
                }
        }

        free(temp);
        return 0;
}
```

```
int permute2(char w[]) {
        char *temp;
        int i, j, k;
        temp = calloc (strlen(w)+1, sizeof(char));

        for (i=0;i<3; i++) {
                for (j = 0; j<3; j++) {
                        for (k=0; k<3; k++) {
                                temp[i] = w[0];
                                if (j != i ) {
                                    temp[j] = w[1];
                                        if(k!= i && k!=j ) {
                                                temp[k] = w[2];
                                                printf("%s\n", temp);
                                        }
                                }
                        }
                }
        }
        free(temp);
        return 0;
}
```

Συγκεντρώνουμε τα for loops

Μερικές ενέργειες εκτελούνται
Περισσότερες φορές

- Γενικεύουμε (για κάθε μήκος αλφαριθμητικού)
  - Χρησιμοποιούμε πίνακα counters για αποθήκευση των τιμών των μετρητών
  - Φτιάχνουμε συνάρτηση next που επιστρέφει την επόμενη τιμή όλων των απαριθμητών των βρόχων
  - Φτιάχνουμε συνάρτηση alldiff που ελέγχει αν όλες οι τιμές είναι διαφορέτικές

```
int permute3(char w[]) {
        char *temp;
        int i ;
        int *counters;
        int count =0;
        temp = calloc (strlen(w)+1, sizeof(char));
        counters = calloc(strlen(w), sizeof(int));

        do {
                if (alldiff(counters, strlen(w)) ) {
                        for(i=0; i<strlen(w); i++)
                                temp[counters[i]] = w[i];
                        printf("%s\n", temp);
                        count++;
                }

        } while (next(counters, strlen(w)));

        printf("A grand total of %d\n", count);
        free(counters);
        free(temp);
        return 0;
}
```

# Ταξινομημένα permutations;

- Όχι πολύ έξυπνος τρόπος:
  - Βάλε τα permutations σε ένα πίνακα
  - Ταξινόμησε με συνάρτηση της βασικής βιβλιοθήκης `qsort()`

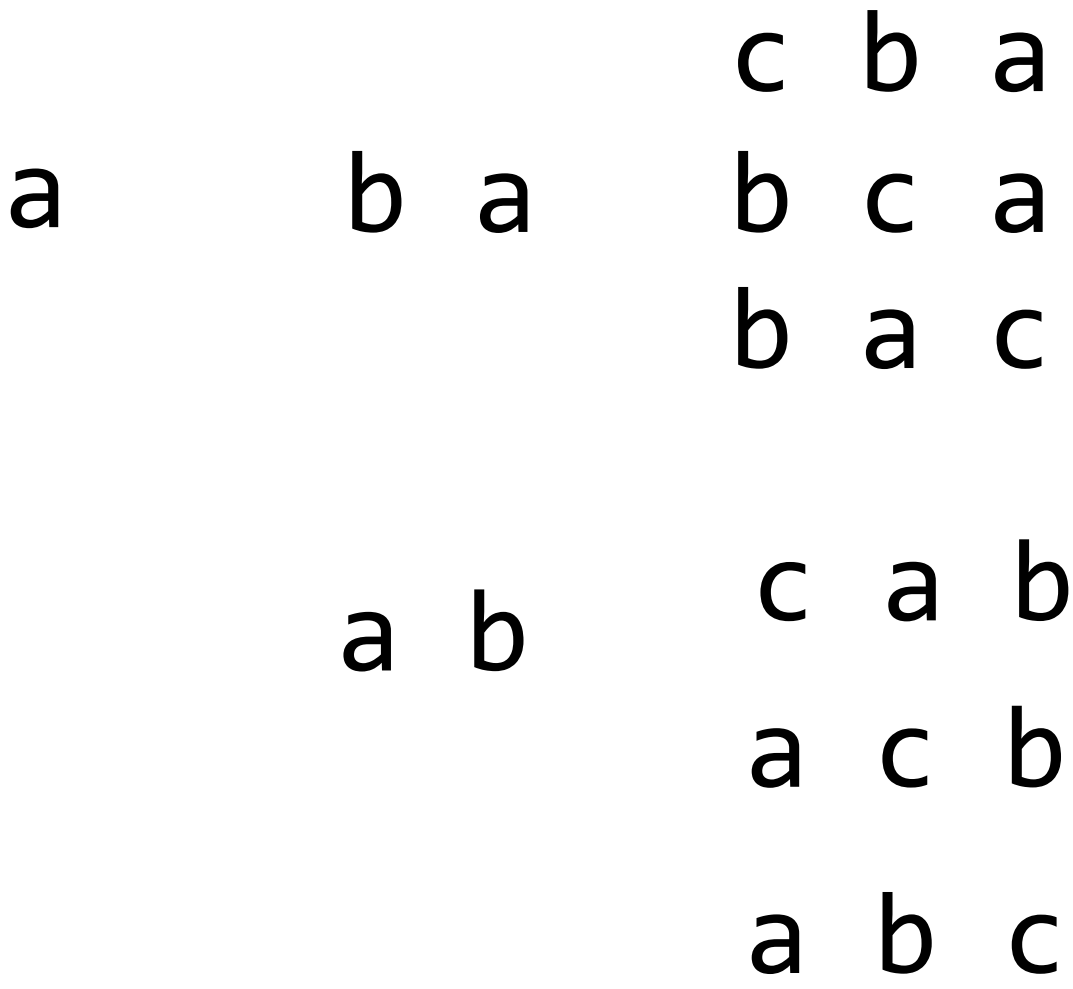
- Τα permutations μπορεί να είναι **πολλά**!

```
int permute4(char w[]) {
        char *temp;
        int i, pos ;
        int *counters;
        int count =0;
        int size = fact(strlen(w));
        int iterations = 0;
        char *perms ;
        perms = calloc( size *(strlen(w)+1) , sizeof (char));
        temp = calloc (strlen(w)+1, sizeof (char));
        counters = calloc(strlen(w), sizeof (int));
        do {
                if (alldiff(counters, strlen(w))) {
                        for (i=0; i<strlen(w); i++)
                                temp[counters[i]] = w[i];
                        printf("%s\n", temp);
                        strcpy( perms + (count * (strlen(w)+1)),  temp);
                        count++;
                }
                iterations ++;
        } while(next(counters, strlen(w)));

        /* continued: next slide */
```

```
 /* continued from previous slide */

 printf("A grand total of %d\n", count);
 printf("Iterations %d\n", iterations);
 qsort(perms, count, strlen(w)+1,
               (int (*)(const void *, const void*)) strcmp);
 printf("sorted:\n");
 for (i=0; i<count; i++)
       printf("%s\n", perms + i*(strlen(w)+1));
 free(perms);
 free(counters);
 free(temp);
 return 0;
}
```

- Άλλος τρόπος παραγωγής permutations.
- Σε ένα permutation με `n-1` χαρακτήρες, παρεμβάλω τον n-οστό χαρακτήρα σε κάθε δυνατή θέση.
- Χρειάζεται να φτιάξω συνάρτηση `insert()` που παρεμβάλει έναν χαρακτήρα σε συγκεκριμένη θέση σε αλφαριθμητικό

# Παρεμβολή (insert) χαρακτήρα



```
int permute5(char w[]){
int c0, c1, c2;
char t[4][10] = {0};

insert(t[0],w[0],0);
for (c0=0; c0<strlen(t[0])+1; c0++) {
        strcpy(t[1],t[0]);
        insert(t[1],w[1],c0);
        for (c1=0; c1<strlen(t[1])+1; c1++){
                strcpy(t[2],t[1]);
                insert(t[2],w[2],c1);
                for (c2=0;c2<strlen(t[2])+1;c2++){
                        strcpy(t[3],t[2]);
                        insert(t[3],w[3],c2);
                        printf("%s\n", t[3]);
                }
        }
 }

 return 0;
}
```

# Τρόπος 1

```
int genpermute5(char w[]){
int c[3]={0};
char t[4][10] = {0};
int j;

 insert(t[0],w[0],0);
 for (c[0]=0;c[0]<strlen(t[0])+1;c[0]++) {
       for (c[1]=0;c[1]<strlen(t[1])+1;c[1]++){
             for (c[2]=0;c[2]<strlen(t[2])+1;c[2]++){
                   strcpy(t[1],t[0]);
                   insert(t[1],w[1],c[0]) ;
                   strcpy(t[2],t[1]);
                   insert(t[2],w[2],c[1]);
                   strcpy(t[3],t[2]);
                   insert(t[3],w[3],c[2]);
                   printf("%s\n", t[3]);
             }
       }
 }
 return 0;
}
```

Βήματα για γενίκευση: πίνακας για μετρητές και συγκεντρώνω τα for

(μερικές ενέργειες γίνονται περισσότερες φορές)

```
int gen2permute5(char w[]){
 int c[3]={0};
 char t[4][10] = {0};
 int j;

 insert(t[0],w[0],0);
 for (c[0]=0;c[0]<1+1;c[0]++) {
       for (c[1]=0;c[1]<2+1;c[1]++){
              for (c[2]=0;c[2]<3+1;c[2]++){
                     for (j=1; j<4; j++) {
                            strcpy(t[j], t[j-1]);
                            insert(t[j], w[j], c[j-1]) ;
                     }
                     printf("%s\n", t[3]);
              }
       }
 }

 return 0;
}
```

Αντικαθιστώ την αλληλουχία των προηγούμενων ζευγών
Με ένα εσωτερικό βρόχο for

```
int gen3permute5(char w[]){
int c[10] = {0};
char t[10][10] = {0};
int j;

  insert(t[0],w[0],0);
  do {
        for (j=1; j<strlen(w); j++) {
               strcpy(t[j],t[j-1]);
               insert(t[j],w[j],c[j-1]) ;
        }
        printf("%s\n", t[j-1]);
 } while (nextvector(c, strlen(w)));

 return 0;
}
```

Κρύβω τον τρόπο αλλαγής των μετρητών σε μια συνάρτηση nextvector

# Τρόπος 2

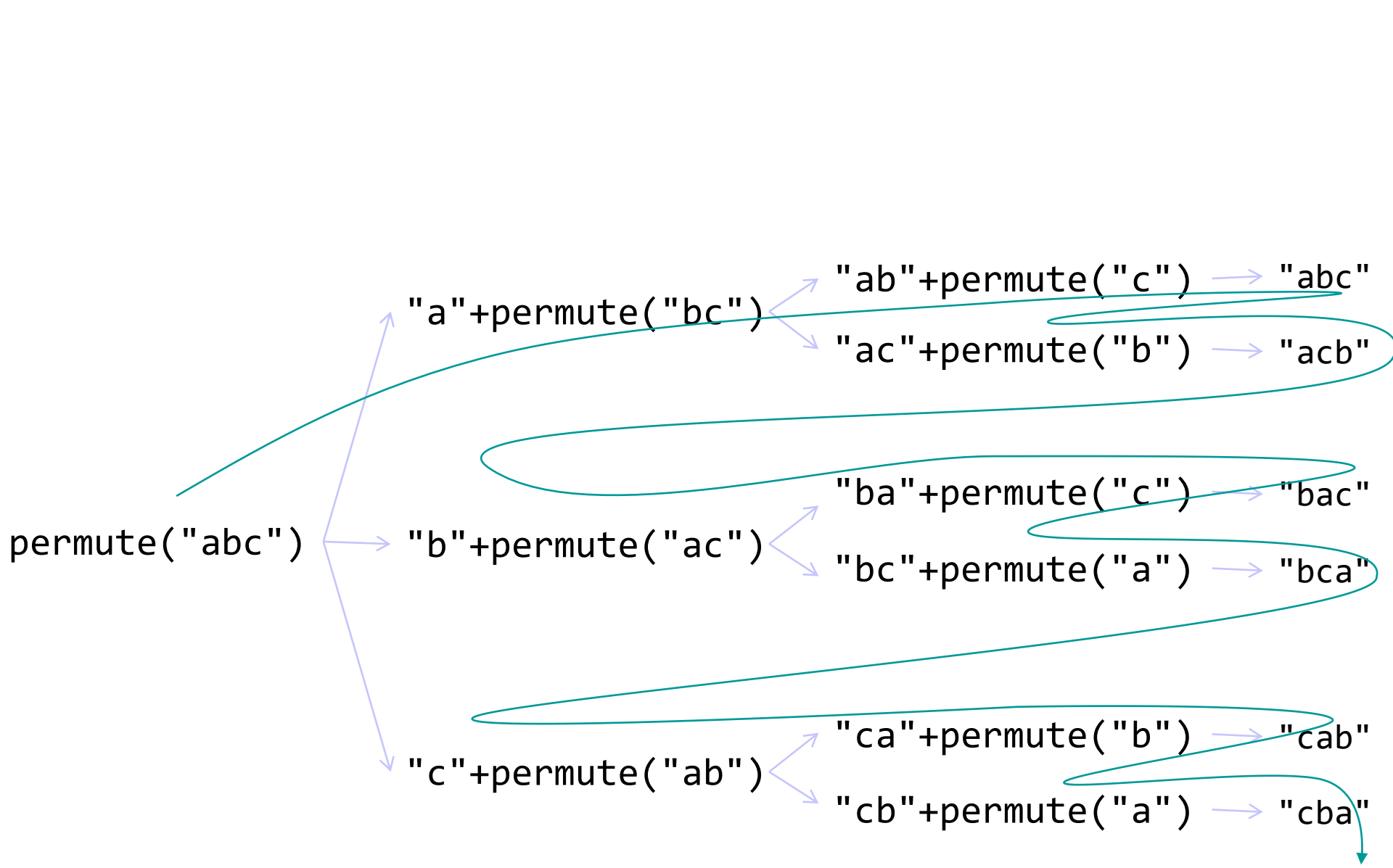
```
int permute5(char w[]){
int c0, c1, c2;
char t[4][10] = {0};

insert(t[0],w[0],0);
for (c0=0; c0<strlen(t[0])+1; c0++) {
        strcpy(t[1],t[0]);
        insert(t[1],w[1],c0);
        for (c1=0; c1<strlen(t[1])+1; c1++){
                strcpy(t[2],t[1]);
                insert(t[2],w[2],c1);
                for (c2=0;c2<strlen(t[2])+1;c2++){
                        strcpy(t[3],t[2]);
                        insert(t[3],w[3],c2);
                        printf("%s\n", t[3]);
                }
        }
 }

 return 0;
}
```

Η δομή του κώδικα κρύβει μια αναδρομική υλοποίηση.

# Αναδρομική υλοποίηση

```
int permute6(char w[],  char t[], int charsleft){
        char tnew[10] = {0};
        int i =0;

        if (charsleft ==0) {
                printf("%s\n", t);
                return 0;
        }

        for (i=0; i<strlen(t)+1; i++) {
                strcpy(tnew, t);
                insert(tnew, w[strlen(w)-charsleft], i);
                permute6(w, tnew, charsleft-1 );
        }

        return 0;
}
```

- Άλλος τρόπος:
  - Πάρε έναν χαρακτήρα και τοποθέτησέ τον στην αρχή
  - Αναδιάταξε τους υπόλοιπους και επικόλλησε το αποτέλεσμα στο τέλος.
- Τα αποτελέσματα προκύπτουν ταξινομημένα
  - Δεν χρειάζεται αποθήκευση όλων
  - Δεν χρειάζεται ταξινόμηση


permute("abc")
"a"+permute("bc")
"b"+permute("ac")
"c"+permute("ab")
"ab"+permute("c")
"abc"
"ac"+permute("b")
"acb"
"ba"+permute("c")
"bac"
"bc"+permute("a")
"bca"
"ca"+permute("b")
"cab"
"cb"+permute("a")
"cba"

```
int permute7( char w[], char prefix[],  char t[], int charsleft){
        char tnew[10] = {0};
        char newprefix[10] ={0};
        char c;
        int i =0;

        if (charsleft == 0){
                printf("%s\n", prefix);
                return 0;
        }
        for (i=0; i<charsleft; i++){
                if (charsleft==strlen(w))
                        strcpy(tnew,w);
                else
                        strcpy(tnew,t);
                c = takeout(tnew, i);
                sprintf(newprefix,"%s%c", prefix, c);
                permute7( w, newprefix, tnew, charsleft-1 );
        }
        return 0;
}
```

Αναδρομική υλοποίηση prefix